NAME

refchan — command handler API of reflected channels

SYNOPSIS

cmdPrefix *option* ?*arg arg ...*?

DESCRIPTION

The Tcl-level handler for a reflected channel has to be a command with subcommands (termed an *ensemble*, as it is a command such as that created by **namespace ensemble** **create**, though the implementation of handlers for reflected channel *is not* tied to **namespace ensemble**s in any way; see **EXAMPLE** below for how to build an **oo::class** that supports the API). Note that *cmdPrefix* is whatever was specified in the call to **chan create**, and may consist of multiple arguments; this will be expanded to multiple words in place of the prefix.

Of all the possible subcommands, the handler *must* support **initialize**, **finalize**, and **watch**. Support for the other subcommands is optional.

MANDATORY SUBCOMMANDS

cmdPrefix **initialize** *channelId mode*
: An invocation of this subcommand will be the first call the *cmdPrefix* will receive for the specified new *channelId*. It is the responsibility of this subcommand to set up any internal data structures required to keep track of the channel and its state.

The return value of the method has to be a list containing the names of all subcommands supported by the *cmdPrefix*. This also tells the Tcl core which version of the API for reflected channels is used by this command handler.

Any error thrown by the method will abort the creation of the channel and no channel will be created. The thrown error will appear as error thrown by **chan create**. Any exception other than an **error** (e.g., **break**, etc.) is treated as (and converted to) an error.

Note: If the creation of the channel was aborted due to failures here, then the **finalize** subcommand will not be called.

The *mode* argument tells the handler whether the channel was opened for reading, writing, or both. It is a list containing any of the strings **read** or **write**. The list may be empty, but will usually contain at least one element.

The subcommand must throw an error if the chosen mode is not supported by the *cmdPrefix*.

cmdPrefix **finalize** *channelId*
An invocation of this subcommand will be the last call the *cmdPrefix* will receive for the specified *channelId*. It will be generated just before the destruction of the data structures of the channel held by the Tcl core. The command handler *must not* access the *channelId* anymore in no way. Upon this subcommand being called, any internal resources allocated to this channel must be cleaned up.

The return value of this subcommand is ignored.

If the subcommand throws an error the command which caused its invocation (usually **chan close**) will appear to have thrown this error. Any exception beyond **error** (e.g., **break**, etc.) is treated as (and converted to) an error.

This subcommand is not invoked if the creation of the channel was aborted during **initialize** (See above).

cmdPrefix **watch** *channelId eventspec*
This subcommand notifies the *cmdPrefix* that the specified *channelId* is interested in the events listed in the *eventspec*. This argument is a list containing any of **read** and **write**. The list may be empty, which signals that the channel does not wish to be notified of any events. In that situation, the handler should disable event generation completely.

Warning: Any return value of the subcommand is ignored. This includes all errors thrown by the subcommand, **break**, **continue**, and custom return codes.

This subcommand interacts with **chan postevent**. Trying to post an event which was not listed in the last call to **watch** will cause **chan postevent** to throw an error.

OPTIONAL SUBCOMMANDS

cmdPrefix **read** *channelId count*
This *optional* subcommand is called when the user requests data from the channel *channelId*. *count* specifies how many *bytes* have been requested. If the subcommand is not supported then it is not possible to read from the channel handled by the command.

The return value of this subcommand is taken as the requested data *bytes*. If the returned data contains more bytes than requested, an error will be signaled and later thrown by the command which performed the read (usually **gets** or **read**). However, returning fewer bytes than requested is acceptable.

Note that returning nothing (0 bytes) is a signal to the higher layers that **EOF** has been reached on the channel. To signal that the channel is out of data right now, but has not yet reached **EOF**, it is necessary to throw the error "EAGAIN", i.e. to either

```
return -code error EAGAIN
```

or

```
error EAGAIN
```

For extensibility any error whose value is a negative integer number will cause the higher layers to set the C-level variable "**errno**" to the absolute value of this number, signaling a system error. However, note that the exact mapping between these error numbers and their meanings is operating system dependent.

For example, while on Linux both

```
return -code error -11
```

and

```
error -11
```

are equivalent to the examples above, using the more readable string "EAGAIN", this is not true for BSD, where the equivalent number is -35.

The symbolic string however is the same across systems, and internally translated to the correct number. No other error value has such a mapping to a symbolic string.

If the subcommand throws any other error, the command which caused its invocation (usually **gets**, or **read**) will appear to have thrown this error. Any exception beyond **error**, (e.g., **break**, etc.) is treated as and converted to an error.

cmdPrefix **write** *channelId data*

This *optional* subcommand is called when the user writes data to the channel *channelId*. The *data* argument contains *bytes*, not characters. Any type of transformation (EOL, encoding) configured for the channel has already been applied at this point. If this subcommand is not supported then it is not possible to write to the channel handled by the command.

The return value of the subcommand is taken as the number of bytes written by the channel. Anything non-numeric will cause an error to be signaled and later thrown by the command which performed the write. A negative value implies that the write failed. Returning a value greater than the number of bytes given to the handler, or zero, is forbidden and will cause the Tcl core to throw an error.

To signal that the channel is not able to accept data for writing right now, it is necessary to throw the error "EAGAIN", i.e. to either

```
return -code error EAGAIN
```

or

```
error EAGAIN
```

For extensibility any error whose value is a negative integer number will cause the higher layers to set the C-level variable "**errno**" to the absolute value of this number, signaling a system error. However, note that the exact mapping between these error numbers and their meanings is operating system dependent.

For example, while on Linux both

```
return -code error -11
```

and

```
error -11
```

are equivalent to the examples above, using the more readable string "EAGAIN", this is not true for BSD, where the equivalent number is -35.

The symbolic string however is the same across systems, and internally translated to the correct number. No other error value has such a mapping to a symbolic string.

If the subcommand throws any other error the command which caused its invocation (usually **puts**) will appear to have thrown this error. Any exception beyond **error** (e.g., **break**, etc.) is treated as and converted to an error.

cmdPrefix **seek** *channelId offset base*

This *optional* subcommand is responsible for the handling of **chan seek** and **chan tell** requests on the channel *channelId*. If it is not supported then seeking will not be possible for the channel.

The *base* argument is the same as the equivalent argument of the builtin **chan seek**, namely:

start
: Seeking is relative to the beginning of the channel.

current
: Seeking is relative to the current seek position.

end
: Seeking is relative to the end of the channel.

The *offset* is an integer number specifying the amount of **bytes** to seek forward or backward. A positive number should seek forward, and a negative number should seek backward. A channel may provide only limited seeking. For example sockets can seek forward, but not backward.

The return value of the subcommand is taken as the (new) location of the channel, counted from the start. This has to be an integer number greater than or equal to zero. If the subcommand throws an error the command which caused its invocation (usually **chan seek**, or **chan tell**) will appear to have thrown this error. Any exception beyond **error** (e.g., **break**, etc.) is treated as and converted to an error.

The offset/base combination of 0/**current** signals a **chan tell** request, i.e., seek nothing relative to the current location, making the new location identical to the current one, which is then returned.

cmdPrefix **configure** *channelId option value*
This *optional* subcommand is for setting the type-specific options of channel *channelId*. The *option* argument indicates the option to be written, and the *value* argument indicates the value to set the option to.

This subcommand will never try to update more than one option at a time; that is behavior implemented in the Tcl channel core.

The return value of the subcommand is ignored.

If the subcommand throws an error the command which performed the (re)configuration or query (usually **fconfigure** or **chan configure**) will appear to have thrown this error. Any exception beyond **error** (e.g., **break**, etc.) is treated as and converted to an error.

cmdPrefix **cget** *channelId option*
This *optional* subcommand is used when reading a single type-specific option of channel *channelId*. If this subcommand is supported then the subcommand **cgetall** must be supported as well.

The subcommand should return the value of the specified *option*.

If the subcommand throws an error, the command which performed the (re)configuration or query (usually **fconfigure** or **chan configure**) will appear to have thrown this error. Any exception beyond *error* (e.g., **break**, etc.) is treated as and converted to an error.

cmdPrefix **cgetall** *channelId*
This *optional* subcommand is used for reading all type-specific options of channel *channelId*. If this subcommand is supported then the subcommand **cget** has to be supported as well.

The subcommand should return a list of all options and their values. This list must have an even number of elements.

If the subcommand throws an error the command which performed the (re)configuration or query (usually **fconfigure** or **chan configure**) will appear to have thrown this error. Any exception beyond **error** (e.g., **break**, etc.) is treated as and converted to an error.

cmdPrefix **blocking** *channelId mode*
This *optional* subcommand handles changes to the blocking mode of the channel *channelId*. The *mode* is a boolean flag. A true value means that the channel has to be set to blocking, and a false value means that the channel should be non-blocking.

The return value of the subcommand is ignored.

If the subcommand throws an error the command which caused its invocation (usually **fconfigure** or **chan configure**) will appear to have thrown this error. Any exception beyond **error** (e.g., **break**, etc.) is treated as and converted to an error.

NOTES

Some of the functions supported in channels defined in Tcl's C interface are not available to channels reflected to the Tcl level.

The function **Tcl_DriverGetHandleProc** is not supported; i.e., reflected channels do not have OS specific handles.

The function **Tcl_DriverHandlerProc** is not supported. This driver function is relevant only for stacked channels, i.e., transformations. Reflected channels are always base channels, not transformations.

The function **Tcl_DriverFlushProc** is not supported. This is because the current generic I/O layer of Tcl does not use this function anywhere at all. Therefore support at the Tcl level makes no sense either. This may be altered in the future (through extending the API defined here and changing its version number) should the function be used at some time in the future.

EXAMPLE

This demonstrates how to make a channel that reads from a string.

```
oo::class create stringchan {
    variable data pos
    constructor {string {encoding {}}} {
        if {$encoding eq ""} {set encoding [encoding system]}
        set data [encoding convertto $encoding $string]
        set pos 0
    }

    method initialize {ch mode} {
        return "initialize finalize watch read seek"
    }
    method finalize {ch} {
        my destroy
    }
    method watch {ch events} {
        # Must be present but we ignore it because we do not
        # post any events
    }

    # Must be present on a readable channel
    method read {ch count} {
        set d [string range $data $pos [expr {$pos+$count-1}]]
        incr pos [string length $d]
        return $d
    }

    # This method is optional, but useful for the example below
    method seek {ch offset base} {
        switch $base {
            start {
                set pos $offset
            }
            current {
                incr pos $offset
            }
            end {
                set pos [string length $data]
                incr pos $offset
            }
        }
        if {$pos < 0} {
            set pos 0
        } elseif {$pos > [string length $data]} {
            set pos [string length $data]
        }
        return $pos
    }
}

# Now we create an instance...
set string "The quick brown fox jumps over the lazy dog.\n"
set ch [chan create read [stringchan new $string]]

puts [gets $ch];   # Prints the whole string

seek $ch -5 end;
puts [read $ch];   # Prints just the last word
```

SEE ALSO

chan, **transchan**

KEYWORDS

API, channel, ensemble, prefix, reflection